G-Level Computer Science Pseudocode Guide

A Companion

A Companion

to

G-Level Computer Science Workbook

G-Level Computer Science Workbook

Created by Diane Anthony

Version 1

October 2016

Table of Contents

ACKNOWLEDGEMENTS

Since nothing is done in isolation of other human beings, either directly or indirectly, I would like to say special thanks to my colleagues within the Computer Science Department for their support in ensuring the successful compilation, reproduction and distribution of this Pseudocode Guide:

INTRODUCTION

This Pseudocode Guide is intended to provide students and teachers with a basic format for writing pseudocode. It is the standard that will be used within SCIE Computer Science Department at G-Level. This guide is aimed at helping students to produce pseudocode in a clear, systematic format that makes their presentation easier for others to understand, modify, upgrade and assess.

It should be treated as a supplement to a Course textbook, course syllabus, workbooks and other relevant resource material.

Note:

To err is human. Hence, please accept my apologies for all organisational inconsistencies in the presentation of concepts and typographical or grammatical errors. If you locate any organisational inconsistencies within the presentation of concepts, typography, spelling and/or grammatical error(s), I would be grateful if you would please email me your observation at diane.anthony@alevel.com.cn citing the specific page(s) of the error(s).

G-LEVEL PSEUDOCODE STRUCTURE

DATA TYPES

INTEGER:	Whole number. E.g. 9, 16
REAL:	A decimal number. E.g. 3.4 or -3.5
CHAR:	One character E.g. 'A'
STRING:	A sequences of blanks, letters or words E.g. "Love", "Today is a lovely day.", " "
BOOLEAN:	Logical values. E.g. TRUE or FALSE
DATE:	Any date in date format. E.g. 03/12/2016

ASSIGNMENT

Assignment means to store the value on the right side of an **assignment operator** into the variable (memory location) on the left side of the assignment operator. The assignment operator is ⟵

Example 1: score ⟵10 *The value **10** is store inside the variable (memory location called "score".*

Example 2: sum ⟵number1 + number2 *The values on the right are added and then stored inside the variable (memory location called "sum"*.

STANDARD ARITHMETIC OPERATOR SYMBOLS ARE USED:

1. + Addition
2. - Subtraction
3. * Multiplication
4. ÷ or / Division - **E.g. 9 / 2 = 4.5 (note:** *Always results in a real value***)**
5. MOD - **E.g. 9 MOD 2 = 1**
6. DIV - **E.g. 9 DIV 2 = 4**

LOGIC OPERATORS

AND - All INPUTS ***must*** be positive to have a positive OUTPUT

Example

0 AND 0 = 0

0 AND 1 = 0

1 AND 0 = 0

1 AND 1 = 1

OR - At least one INPUT ***must*** be positive to have a positive OUTPUT

Example

0 OR 0 = 0

0 OR 1 = 1

1 OR 0 = 1

1 OR 1 = 1

NOT - The OUTPUT is always the reverse of the INPUT.

Example

NOT 0 = 1

NOT 1 = 0

IDENTIFIERS

An identifier is the name given to a variable, constant or subroutine

1. It should properly describe the variable it refers to
2. It should ONLY begin with a letter or underscore.
3. It may include an underscore but NO other symbol
4. Simple variables may be identified with mixed cased letters. E.g. studentAge
5. It may NOT include any spaces.
6. Python's reserved words or keywords cannot be used as identifiers. They have specific meanings to Python, such as ***if*** and ***while***
7. Identifiers are case-sensitive. 'number' is different from 'NUMBER' and 'Number'.

Note: A constant generally has a fixed value. E.g. The value of pi.

COMMENTS

Comments should follow double back-slashes (//) and be placed at the side of statements and at above small sections of code.

Example: //This code is used to prevent blank entries

INPUT and OUTPUT STATEMENTS

Input and Output statements should be written in uppercase.

Example

INPUT <Identifier> //Inputs one piece of data

INPUT <Identifier1>, <Identifier2> //Inputs two pieces of data

READ <Identifier> //Inputs one piece of data

READ <Identifier1>, <Identifier2> //Inputs two pieces of data

Example 1

OUTPUT <Identifier> //Outputs one piece of data

OUTPUT <Identifier1>, <Identifier2> //Outputs two pieces of data

WRITE <Identifier> //Outputs one piece of data

WRITE <Identifier1>, <Identifier2> //Outputs two pieces of data

Example 2

OUTPUT <statement>

OUTPUT < statement >

WRITE < statement >

WRITE < statement >

CONDITIONAL STATEMENTS

Single condition

```
IF <condition> THEN
    <statement>
ENDIF
```

Two conditions

```
IF <condition> THEN
    <statement>
ELSE
    <statement>
ENDIF
```

More than two conditions

```
IF <condition> THEN
    <statement>
ELSEIF <condition> THEN
    <statement>
ELSE
    <statement>
ENDIF
```

Nested-IF– (If within an if)

```
IF <condition> THEN
        IF <condition> THEN
                <statement>
        ENDIF
ELSE
        IF <condition> THEN
                <statement>
        ENDIF
ENDIF
```

CASE

```
CASE OF <identifier>
        <value 1>: <statement>
        <value 2>: <statement>
        ...
ENDCASE
```

```
CASE OF <identifier>
        <value 1>: <statement>
        <value 2>: <statement>
        ...
        OTHERWISE <statement> //Used if identifier is not selected
ENDCASE
```

Example 1

```
INPUT Choice
CASE OF Choice
        1: “Ice cream”
        2: “Pizza”
        3: “Bubble Tea”
        4: “Sichuan Noodles”
ENDCASE
```

Example 2

```
INPUT Choice
CASE OF Choice
        1: “Ice cream”
        2: “Pizza”
        3: “Bubble Tea”
        4: “Sichuan Noodles”
        OTHERWISE: “Don’t have your selected Choice”
ENDCASE
```

ITERATIVE STATEMENTS

Pre-condition

Pre-condition statements test a condition first and ONLY executes the statements within the loop if the condition is TRUE.

```
WHILE <condition> DO (pre-condition)
        <statements>
ENDWHILE
```

Example 1

```
WHILE BankCard is not present DO
        OUTPUT “Input Bank Care”
ENDWHILE
```

Post-condition

```
REPEAT
    <Statements>
UNTIL <condition>
```

Example 2

```
REPEAT
        OUTPUT "Enter PIN"
UNTIL PIN = 123
```

FOR…NEXT

```
FOR <identifier> ← <value1> TO <value2>
        <statements>
NEXT
```

Note: *Identifiers and Values MUST evaluate to integers*

Example 1

```
FOR count← 1 TO 10
        OUTPUT “hello”
NEXT
```

Difference between WHILE…DO and FOR..NEXT statements

WHILE…DO is generally used when the number of repetitions is unknown

FOR…NEXT is used when the number of repetitions is known.

ARRAYS – 1 Dimensional

DECLARE <identifier>: ARRAY[<l>:<n>] OF <data type>//Declare an array

Example 1

DECLARE TeamScores: ARRAY[1:3] OF INTEGER

Example 2 – Assigning values to arrays

TeamNames[1] ← **"Jaguars"** *“Jaguar” is stored at position 1 in the Array.*

TeamNames [2] ← **'D'** *The value ‘D’ is stored at position 2 in the Array.*

G2Students← **G1Students** *The Array on the right is copied to the Array on the left. Arrays should have the same data type and size.*

Example 3

Teams [1 TO 3] ← " " not allowed in pseudocode but works in program code

```
//The following should be used instead
FOR Score = 1 TO 3
    Teams[Score] ← " "
NEXT Index
```

SUBROUTINES

PROCEDURE – no parameters

```
PROCEDURE <identifier>
    <statements>
ENDPROCEDURE
```

PROCEDURE – with parameters

```
PROCEDURE <identifier>(<p1>:<data type>,<p2>:<data type>...)
    <statements>
ENDPROCEDURE
```

PROCEDURE CALL – no parameters

```
CALL <identifier>
```

PROCEDURE CALL – with parameters

```
CALL <identifier>(Value1,Value2)
```

Example 1 – Procedure with no parameters

```
PROCEDURE Addition
    INPUT number1, number2
    Sum = number1 + number2
    OUTPUT Sum
ENDPROCEDURE
CALL Addition
```

Example 2 – Procedure with parameters

```
PROCEDURE Addition(number1: integer, number2:integer)
    Sum = number1 + number2
    OUTPUT Sum
ENDPROCEDURE

CALL Addition(number1, number2)
```

Note: Procedure calls are complete statements

FUNCTIONS– no parameters

```
FUNCTION <identifier> RETURNS <data type>
    <statements>
ENDFUNCTION
```

FUNCTIONS– with parameters

```
FUNCTION <identifier>(<p1>:<data type>,<p2>:<data type>.) RETURNS <data type>
    <statements>
ENDFUNCTION
```

Example 1 – Function with no parameters

```
FUNCTION Min() RETURNS INTEGER
    INPUT Number1, Number2
    IF Number1 < Number2 THEN
        RETURN Number1
    ELSE
        RETURN Number2
    ENDIF
ENDFUNCTION
OUTPUT "The smaller number is  ", Min()
```

Example 2 – Function with parameters

```
FUNCTION Min(Number1:INTEGER, Number2:INTEGER) RETURNS
INTEGER
    IF Number1 < Number2 THEN
        RETURN Number1
    ELSE
        RETURN Number2
    ENDIF
ENDFUNCTION
OUTPUT "The smaller number is  ", Min(3, 6)
```

Note: Function calls are **NOT** complete statements. Functions return a value that is then used as part of an expression.

SOURCES

Cambridge IGCSE Pseudocode Guide for Teachers for examination in 2017 – Cambridge IGCSE Computer Science 0478